Network Constructors: A Model for Programmable Matter*

Othon Michail[1,2] and Paul G. Spirakis[1,2]

[1] Department of Computer Science, University of Liverpool, Liverpool, UK
[2] Computer Technology Institute and Press "Diophantus" (CTI), Patras, Greece
Email: Othon.Michail@liverpool.ac.uk, P.Spirakis@liverpool.ac.uk

Abstract. We discuss recent theoretical models for programmable matter operating in a dynamic environment. In the basic *Network Constructors* model, all devices are finite automata, begin from the same initial state, execute the same protocol, and can only interact in pairs. The interactions are scheduled by a *fair* (or uniform random) scheduler, in the spirit of *Population Protocols*. When two devices interact, the protocol takes as input their states and the state of the connection between them (*on*/*off*) and updates all of them. Initially all connections are *off*. The goal of such protocols is to eventually construct a desired stable network, induced by the edges that are *on*. We present protocols and lower bounds for several basic network construction problems and also universality results. We next highlight minimal strengthenings of the model, that can be exploited by appropriate network-transformation protocols in order to achieve *termination* and the *maximum computational power* that one can hope for in this family of models. Finally, we discuss a more applied version of these abstract models, enriched with geometric constraints, aiming at capturing some first physical restrictions in potential future programmable matter systems operating in dynamic environments.

1 Introduction

The realization of computing systems and computer networks was indisputably one of the most outstanding achievements of science and engineering of the last century. The impact of Information and Communication Technologies on society, industry, and everyday life was incomparable. Digital communications and the Internet have made the world look much smaller, personal computers radically changed office work, largely simplifying it, high processing speeds made it possible for the first time to simulate and accurately predict a wide range of physical phenomena, from weather forecast to chemical reactions and whole-cell simulations [KSM+12], and combined to increased storage capabilities, transformed the world of paper to a world of digital information, where everything, from a datatrace of successful collisions in CERN that produced the Higgs boson [CKS+12]

* Supported in part by the School of EEE/CS of the University of Liverpool, NeST initiative, and the EU IP FET-Proactive project MULTIPLEX under contract no 317532.

to the human genome, can be stored and retrieved. Computing and Information Sciences have been extremely successful in revealing the laws underlying all possible ways of manipulating information. Every possible object, system or problem can be encoded in an appropriate binary representation, which can then be stored, processed, retrieved and transmitted. It would be reasonable to say that the 20th century was *the century of information.*

However, the story does not seem to end here. The established knowledge of manipulating information seems to have opened the road towards a vision that will further reshape society to an unprecedented degree. This vision concerns our ability to *manipulate matter* via information-theoretic and computing mechanisms and principles. It will be the jump from amorphous information to the *incorporation of information to the physical world.* Information will not only be part of the physical environment: it will constantly interact with the surrounding environment and will have the ability to reshape it. *Matter will become programmable* [GCM05] which is a plausible future outcome of progress in high-volume nanoscale assembly that makes it feasible to inexpensively produce millimeter-scale units that integrate computing, sensing, actuation, and locomotion mechanisms. This will enable the astonishing possibility of transferring the discrete dynamics from the computer memory black-box to the real world and to achieve a *physical realization of any computer-generated object.* "It will have profound implications for how we think about chemistry and materials. Materials will become user-programmed and smart, adapting to changing conditions in order to maintain, optimize or even create a whole new functionality using means that are intrinsic to the material itself. It will even change the way we think about engineering and manufacturing. We will for the first time be capable of building smart machines that adapt to their surroundings, such as an airplane wing that adjusts its surface properties in reaction to environmental variables" [Zak07], or even further realize machines that can self-built autonomously.

This vision is not a human invention. It is an inspiration from a property that pervades the biological world. Every biological organism is a collection of relatively simple units of matter (the cells) coupled with information storing, processing, and transmission capabilities. Moreover, the effort to realize this vision has already begun and the first outcomes are more than promising. For example, it has been already demonstrated that it is possible to fold long, single-stranded DNA molecules into arbitrary nanoscale two-dimensional shapes and patterns [Rot06]. Also, a system was recently reported that demonstrates programmable self-assembly of complex two-dimensional shapes with a thousand-robot swarm [RCN14]. "This was enabled by creating small, cheap, and simple autonomous robots designed to operate in large groups and to cooperate through local interactions and by developing a collective algorithm for shape formation that is highly robust to the variability and error characteristic of large-scale decentralized systems" [RCN14]. Other systems for programmable matter include the Robot Pebbles [GKR10], consisting of 1cm cubic programmable matter modules able to form 2-dimensional (abbreviated "2D" throughout) shapes through self-disassembly, and the Millimotein [KCL$^+$12], a chain of programmable matter

which can fold itself into digitized approximations of arbitrary 3-dimensional (abbreviated "3D" throughout) shapes.

Apart from the fact that systems work is still in its infancy, there is also an apparent lack of unifying formalism and theoretical treatment. The following are some of the very few exceptions aiming at understanding the fundamental possibilities and limitations of this prospective. The area of *algorithmic self-assembly* tries to understand how to program molecules (mainly DNA strands) to manipulate themselves, grow into machines and at the same time control their own growth [Dot12]. The theoretical model guiding the study in algorithmic self-assembly is the Abstract Tile Assembly Model (aTAM) [Win98, RW00] and variations. Recently, a model, called the *nubot* model, was proposed for studying the complexity of self-assembled structures with active molecular components [WCG$^+$13]. This model "is inspired by biology's fantastic ability to assemble biomolecules that form systems with complicated structure and dynamics, from molecular motors that walk on rigid tracks and proteins that dynamically alter the structure of the cell during mitosis, to embryonic development where large-scale complicated organisms efficiently grow from a single cell" [WCG$^+$13]. Another very recent model, called the *Network Constructors* model, studied what stable networks can be constructed by a population of finite-automata that interact randomly like molecules in a well-mixed solution and can establish bonds with each other according to the rules of a common small protocol [MS16b]. Interestingly, the special case of the model that cannot create bonds (known as the *Population Protocol* model [AAD$^+$06]) is known to be formally equivalent to *chemical reaction networks* (CRNs), which model chemistry in a *well-mixed solution* and are widely used to describe information processing occurring in natural cellular regulatory networks [Dot14]. Also the recently proposed *Amoebot* model, offers a versatile framework to model self-organizing particles and facilitates rigorous algorithmic research in the area of programmable matter [DDG$^+$14, DGP$^+$16].

At the same time, recent research in distributed computing theory and practice is taking its first timid steps on the pioneering endeavor of investigating the possible *relationships of distributed computing systems to physical and biological systems*. The first main motivation for this is the fact that a wide range of physical and biological systems are governed by underlying laws that are essentially *algorithmic*. The second is that the higher-level physical or behavioral properties of such systems are usually the outcome of the coexistence, which may include both cooperation and competition, and constant interaction of *very large numbers of relatively simple distributed entities* respecting such laws. This effort, to the extent that its perspective allows, is expected to promote our understanding on the algorithmic aspects of our (distributed) natural world and to develop innovative artificial systems inspired by them.

In the present paper, we shall focus on the Network Constructors model and its existing variations. In Section 2, we present the basic Network Constructors model and give the main definitions to be used in the sequel. In Section 3, we present protocols for the spanning line construction problem and bounds

for other basic network construction problems. Section 4 goes one step further, showing how one can establish universality results. In Section 5, we show how network-transformation protocols can exploit minimal strengthenings of the basic model, in order to maximize the computational power. Section 6 discusses a geometric variant of the basic model, in which the nodes can be programmed to self-assemble into complex 2D or 3D shapes. Finally, Section 7 highlights some promising directions for future research.

2 The Network Constructors Model

Suppose a set of tiny computational devices (possibly at the nanoscale) are injected into a human circulatory system for the purpose of monitoring or even treating a disease. The devices are incapable of controlling their mobility. The mobility of the devices, and consequently the interactions between them, stems solely from the dynamicity of the environment, the blood flow inside the circulatory system in this case. Additionally, each device alone is incapable of performing any useful computation, as the small scale of the device highly constrains its computational capabilities. The goal is for the devices to accomplish their task via cooperation. To this end, the devices are equipped with a mechanism that allows them to create bonds with other devices (mimicking nature's ability to do so). So, whenever two devices come sufficiently close to each other and interact, apart from updating their local states, they may also become connected by establishing a physical connection between them. Moreover, two connected devices may at some point choose to drop their connection. In this manner, the devices can organize themselves into a desired global structure. This network-constructing self-assembly capability allows the artificial population of devices to evolve greater complexity, better storage capacity, and to adapt and optimize its performance to the needs of the specific task to be accomplished.

Our goal in [MS16b] was to study the fundamental problem of *network construction* by a distributed computing system. The system consists of a set of n processes that are capable of performing local computation (via pairwise interactions) and of forming and deleting connections between them. Connections between processes can be either *physical* or *virtual* depending on the application. In the most general case, a connection between two processes can be in one of a finite number of possible states. For example, state 0 could mean that the connection does not exist while state $i \in \{1, 2, \ldots, k\}$, for some finite k, that the connection exists and has strength i. We considered the simplest case, which we call the *on/off* case, in which, at any time, a connection can either exist or not exist; that is, there are just two states for the connections, 1 and 0, respectively. If a connection exists we also say that it is *active* and if it does not exist we say that it is *inactive*. Initially all connections are inactive and the goal is for the processes, after interacting and activating/deactivating connections for a while, to end up with a desired *stable network*. In the simplest case, the output-network is the one induced by the active connections and it is stable when no connection changes state any more.

Our aim in [MS16b] was to initiate this study by proposing and studying a very *simple*, yet sufficiently generic, model for distributed network construction. To this end, we assumed the computationally weakest type of processes. In particular, the processes are finite automata that all begin from the same initial state and all execute the same finite program which is stored in their memory (i.e., the system is *homogeneous*). The communication model that we considered is also very minimal. In particular, we considered processes that are inhabitants of an *adversarial environment* that has total control over the interprocess interactions. Such an environment is modeled by an adversary scheduler that operates in discrete steps, selecting in every step a pair of processes which then interact according to the common program. This represents very well systems of (not necessarily computational) entities that interact in pairs whenever two of them come sufficiently close to each other. When two processes interact, the program takes as input the states of the interacting processes and the state of their connection and outputs a new state for each process and a new state for the connection. The only restriction that we imposed on the scheduler, in order to study the constructive power of the model, is that it is *fair*, by which we mean the weak requirement that, at every step, it assigns to every reachable configuration of the system a non-zero probability to occur. In other words, a fair scheduler cannot forever conceal an always reachable configuration of the system. Note that under such a generic scheduler, we cannot bound the running time of our constructors. To estimate the efficiency of our solutions, we assume a *uniform random scheduler*, one of the simplest fair probabilistic schedulers. The uniform random scheduler selects in every step independently and uniformly at random a pair of processes to interact from all such pairs. What renders this model interesting is, as we shall see, its ability to achieve complex global behavior via a set of notably simple, uniform (i.e., with codes that are independent of the size of the system), homogeneous, and cooperative entities.

We now give a simple illustration of the above. Assume a set of n very weak processes that can only be in one of two states, "black" or "red". Initially, all processes are black. We can think of the processes as small particles that move randomly in a fair solution. The particles are capable of forming and deleting physical connections between them, by which we mean that, whenever two particles interact, they can read and write the state of their connection. To keep this first model as simple as possible, we assume that fairness of the solution is independent of the states of the connections. [3] In particular, we assume, for the time being, that, throughout the execution, every pair of processes may be selected for interaction.

Consider now the following simple problem. We want to identically program the initially disorganized particles so that they become self-organized into a *spanning star*. In particular, we want to end up with a unique black particle

[3] This is in contrast to schedulers that would take into account the geometry of the active connections and would, for example, forbid two non-neighboring particles of the same component to interact with each other. Such a geometrically restricted variant, studied in [Mic15], shall be discussed in Section 6.

connected (via active connections) to $n-1$ red particles and all other connections (between red particles) being inactive. Conversely, given a (possibly physical) system that tends to form a spanning star we would like to unveil the code behind this behavior.

Consider the following program. When two black particles that are not connected interact, they become connected and one of them becomes red. When two connected red particles interact they become disconnected (i.e., reds repel). Finally, when a black and a red that are not connected interact they become connected (i.e., blacks and reds attract).

The protocol forms a spanning star as follows. As whenever two blacks interact only one survives and the other becomes red, eventually a unique black will remain and all other particles will be red (we say "eventually", meaning "in finite time", because we do not know how much time it will take for all blacks to meet each other, but, from fairness, we know that this has to occur in a finite number of steps). As blacks and reds attract while reds repel, it is clear that eventually the unique black will be connected to all reds while every pair of reds will be disconnected. Moreover, no rule of the program can modify such a configuration, so the constructed spanning star is stable (see Figure 1). It is worth noting that this very simple protocol is optimal both with respect to (abbreviated "w.r.t." throughout) the number of states that it uses and w.r.t. the time it takes to construct a stable spanning star under the uniform random scheduler.

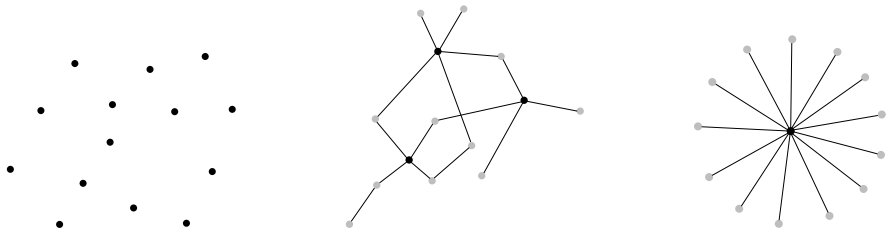

Fig. 1. (a) Initially all particles are black and no active connections exist. (b) After a while, only 3 black particles have survived each having a set of red neighbors (red particles appear as gray here). Note that some red particles are also connected to red particles. The tendency is for the red particles to repel red particles and attract black particles. (c) A unique black has survived, it has attracted all red particles, and all connections between red particles have been deactivated. The construction is a stable spanning star.

Our model for network construction has been strongly inspired by the Population Protocol model [AAD$^+$06] and the Mediated Population Protocol model [MCS11]. In the former, connections do not have states. States on the connections were first introduced in the latter. The main difference to the present model is

that *in those models the focus was on the computation of functions of some input values and not on network construction.* Another important difference is that we now allow the edges to choose between *only two possible states* which was not the case in [MCS11]. As already mentioned, when operating under a uniform random scheduler, population protocols are formally equivalent to *chemical reaction networks* (CRNs). "With upcoming advances in synthetic biology, CRNs are a promising programming language for the design of artificial molecular control circuitry" [Dot14]. However, CRNs and population protocols can only capture the dynamics of molecular counts and not of structure formation. Our model then may be also viewed as an extension of population protocols and CRNs aiming to capture the stable structures that may occur in a well-mixed solution. From this perspective, our goal is to determine what stable structures can result in such systems (natural or artificial), how fast, and under what conditions (e.g., by what underlying codes/reaction-rules).

2.1 Definitions

Definition 1. *A* Network Constructor *(NET) is a distributed protocol defined by a 4-tuple* $(Q, q_0, Q_{out}, \delta)$*, where* Q *is a finite set of* node-states*,* $q_0 \in Q$ *is the* initial node-state*,* $Q_{out} \subseteq Q$ *is the set of* output node-states*, and* $\delta : Q \times Q \times \{0,1\} \to Q \times Q \times \{0,1\}$ *is the* transition function*.*

The system consists of a population V_I of n distributed *processes/nodes*. In the generic case, there is an underlying *interaction graph* $G_I = (V_I, E_I)$ specifying the permissible interactions between the nodes. Interactions are always pairwise. In the basic model, G_I is a *complete undirected interaction graph*, i.e., $E_I = \{uv : u, v \in V_I \text{ and } u \neq v\}$, where $uv = \{u, v\}$. Initially, all nodes in V_I are in the initial node-state q_0. A central assumption of the model is that edges have binary states. An edge in state 0 is said to be *inactive* while an edge in state 1 is said to be *active*. All edges are initially inactive.

Execution of the protocol proceeds in discrete steps. In every step, a pair of nodes uv from E_I is selected by an *adversary scheduler* and these nodes interact and update their states and the state of the edge joining them according to the transition function δ.

A *configuration* is a mapping $C : V_I \cup E_I \to Q \cup \{0,1\}$ specifying the state of each node and each edge of the interaction graph. An *execution* is a finite or infinite sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$ ('$\to$' meaning "goes via a single interaction to"), for all $i \geq 0$. A *fairness condition* is imposed on the adversary to ensure the protocol makes progress. An infinite execution is *fair* if for every pair of configurations C and C' such that $C \to C'$, if C occurs infinitely often in the execution then so does C'. In what follows, every execution of a NET will by definition considered to be fair.

Whenever we study the running time (counted in number of sequential interactions) of a NET, we assume that interactions are chosen by a *uniform random scheduler* which, in every step, selects independently and uniformly at random

one of the $|E_I| = n(n-1)/2$ possible interactions. In this case, the running time becomes a random variable (abbreviated "r.v." throughout) X and our goal is to obtain bounds on the expectation $\mathrm{E}[X]$ of X. Note that the uniform random scheduler is fair with probability 1. We say that an execution of a NET on n processes *constructs a graph* (or *network*) G, if its output stabilizes to a graph isomorphic to G. We say that a NET $\mathcal{A}$ *constructs a graph language* L *with useful space* $g(n) \leq n$, if $g(n)$ is the greatest function for which: (i) for all n, every execution of $\mathcal{A}$ on n processes constructs a $G \in L$ of order at least $g(n)$ (provided that such a G exists) and, additionally, (ii) for all $G \in L$ there is an execution of $\mathcal{A}$ on n processes, for some n satisfying $|V(G)| \geq g(n)$, that constructs G. Equivalently, we say that $\mathcal{A}$ *constructs* L *with waste* $n - g(n)$. Define $\mathbf{REL}(g(n))$ to be the class of all graph languages that are constructible with useful space $g(n)$ by a NET. We call $\mathbf{REL}(\cdot)$ the *relation* or *on/off* class. Also define $\mathbf{PREL}(g(n))$ in precisely the same way as $\mathbf{REL}(g(n))$ but in the extension of the above model in which every pair of processes is capable of tossing an unbiased coin during an interaction between them. In this case, we additionally require that all graphs have the same probability to be constructed by the protocol. We denote by $\mathbf{DGS}(f(l))$ (for "Deterministic Graph Space") the class of all graph languages that are decidable by a Turing Machine (abbreviated "TM" throughout) of (*binary*) space $f(l)$, where l is the length of the adjacency matrix encoding of the input graph.

3 Basic Constructors

Probably the most fundamental network-construction problem, is the problem of constructing a spanning line, i.e., a connected graph in which 2 nodes have degree 1 and $n-2$ nodes have degree 2. Its importance lies in the fact that a spanning line provides an ordering on the processes which can then be exploited (as discussed in Section 4) to simulate a TM and, in this way, to establish universality of the model.

We begin with a lower bound on the expected time required by any NET to construct a spanning line.

Theorem 1 (Line Lower Bound [MS16b]). *The expected time to convergence of any protocol that constructs a spanning line is* $\Omega(n^2)$.

Take any protocol $\mathcal{A}$ that constructs a spanning line and any execution of $\mathcal{A}$ on n nodes. It suffices to show that any execution necessarily passes through a "bottleneck" transition, by which we mean a transition that requires $\Omega(n^2)$ expected number of steps to occur. Observe that, in any execution, the set of active edges eventually stabilizes (in this case, to a spanning line), which implies that there is always a *last* activation/deactivation of an edge. The idea is to focus on this last operation before stabilization, and show that either this operation is a bottleneck transition or an immediately previous operation is a bottleneck transition. In both cases, any execution passes through a bottleneck transition, thus paying at that point an $\Omega(n^2)$ expected number of steps. Indeed, if the last

modification was an activation, then the construction just before this modification was either a line on $n-1$ nodes and an isolated node or two disjoint lines spanning all nodes. In both cases, the expected number of steps until the last edge becomes activated is $\Omega(n^2)$. On the other hand, if the last modification was a deactivation, then this implies that the construction just before this modification was a spanning line with an additional active edge between two nodes, u and v, that are not neighbors on the line. The only interesting case is the one in which the construction was actually a spanning ring. Then, by considering the last modification of an edge that resulted in the ring, we obtain again an expected number of $\Omega(n^2)$ interactions.

We present now our simplest protocol for the spanning line problem.

Simple-Global-Line. $Q = \{q_0, q_1, q_2, l, w\}$, δ: $(q_0, q_0, 0) \to (q_1, l, 1)$, $(l, q_0, 0) \to (q_2, l, 1)$, $(l, l, 0) \to (q_2, w, 1)$, $(w, q_2, 1) \to (q_2, w, 1)$, $(w, q_1, 1) \to (q_2, l, 1)$.

In the initial configuration C_0, all nodes are in state q_0 and all edges are inactive, i.e., in state 0. Every configuration C that is reachable from C_0 consists of a collection of lines and isolated nodes. Additionally, every line has a unique leader which either occupies an endpoint and is in state l or occupies an internal node, is in state w, and moves randomly along the line. Lines can expand towards isolated nodes and two lines can connect their endpoints to get merged into a single line (with total length equal to the sum of the lengths of the merged lines plus one). Both of these operations only take place when the corresponding endpoint of every line that takes part in the operation is in state l. A line resulting from merging, has a w internal-leader and only waits until the random walk of w reaches one endpoint and becomes an l leader. Figure 2 gives an illustration of a typical configuration of the protocol.

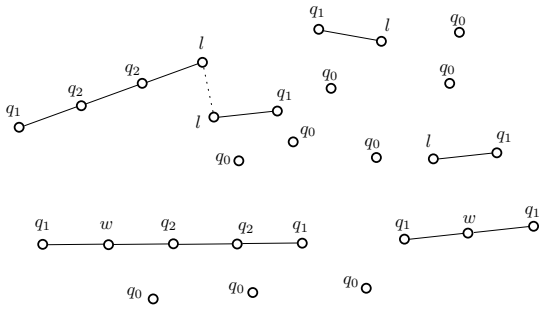

Fig. 2. A typical configuration of *Simple-Global-Line* (after some time has passed).

Theorem 2 ([MS16b]). *Protocol* Simple-Global-Line *constructs a spanning line. It uses 5 states and its expected running time is* $\Omega(n^4)$ *and* $O(n^5)$.

For correctness, we have to prove two things: (i) there is a set $\mathcal{S}$ of output-stable configurations whose active network is a spanning line, (ii) for every reachable configuration C it holds that $C \rightsquigarrow C_s$ ('$\rightsquigarrow$' meaning "goes in one or more steps to") for some $C_s \in \mathcal{S}$.

For the running time upper bound, we have an expected number of $O(n^2)$ steps until progress is made (i.e., for another merging to occur given that at least two l-leaders exist) and $O(n^4)$ steps for the resulting random walk (walk of state w until it reaches one endpoint of the line) to finish and to have the system again ready for progress. This is because the state actually walks only if it interacts with one of its (at most) two neighbors on the line. As only 2 interactions over the $\Theta(n^2)$ possible interactions allow the state to walk, the otherwise $O(n^2)$-time walk is delayed by a factor of $O(n^2)$. As progress must be made $n-2$ times, we conclude that the expected running time of the protocol is bounded from above by $(n-2)[O(n^2)+O(n^4)] = O(n^5)$.

Next, it can be proved that we cannot hope to improve the upper bound on the expected running time by a better analysis by more than a factor of n. For this, we can prove by a Chernoff bound, that the protocol with high probability (abbreviated "w.h.p." throughout) constructs $\Theta(n)$ disjoint lines of length 1 during its course. A set of k disjoint lines implies that $k-1 = \Theta(n)$ distinct merging processes have to be executed in order to merge them all into a common line and each single merging results in the execution of another random walk. Let t_{min} be the first time at which there is a line L of length $h \geq k/4$. It holds that $k/4 \leq h \leq k/2-1$, so there is a remaining length of at least $k-h \geq k-(k/2-1) = k/2+1$ to get merged to L via distinct sequential mergings. Now, if d_i denotes the length of the ith line merged to L, Y the r.v. of the duration of all random walks, and Y_i the r.v. of the duration of the i-th random walk, we have $\mathrm{E}[Y] = \mathrm{E}[\sum_{i=1}^{j} Y_i] = \sum_{i=1}^{j} \mathrm{E}[Y_i] = \sum_{i=1}^{j} n^2(h+d_1+\ldots+d_{i-1})d_i \geq n^2 \sum_{i=1}^{j} h d_i = n^2 h \sum_{i=1}^{j} d_i \geq n^2 \cdot (k/4) \cdot (k/2+1) = n^2 \cdot \Theta(n) \cdot \Theta(n) = \Theta(n^4)$. This proves the desired $\Omega(n^4)$ lower bound.

By using more states, we can develop an alternative protocol that constructs a spanning line much faster. The main difference between this and the previous protocol is that we now totally avoid mergings as they seem to consume much time. As before, when the leaders of two lines interact, one of them becomes eliminated and the edge is activated. But now, the leader that has survived does not initiate a merging process. Instead, it steals a node from the eliminated leader's line and disconnects the two new lines: its own line, which has increased by one and is called *awake*, and the eliminated leader's line, which has decreased by one and is called *sleeping*. The code follows:

Fast-Global-Line. $Q = \{q_0, q_1, q_2, q_2', l, l', l'', f_0, f_1\}$, δ: $(q_0,q_0,0) \to (q_1,l,1)$, $(l,q_0,0) \to (q_2,l,1)$, $(l,l,0) \to (q_2',l',1)$, $(l',q_2,1) \to (l'',f_1,0)$, $(l',q_1,1) \to (l'',f_0,0)$, $(l'',q_2',1) \to (l,q_2,1)$, $(l,f_0,0) \to (q_2,l,1)$, $(l,f_1,0) \to (q_2',l',1)$.

In more detail, when two lines L_1 and L_2 interact via their l-leader endpoints, one of the leaders, say w.l.o.g. that of L_2, becomes l' and the other becomes q_2'.

We can interpret this operation as expanding L_1 on the endpoint of L_2 and obtaining two new lines (still attached to each other): L'_1 which is awake and L'_2 which is sleeping. Now, the l'-leader of L'_1 waits to interact with its neighbor from L'_2 (which is either a q_2 or a q_1) to deactivate the edge between them and disconnect L'_1 from L'_2. This operation leaves L'_1 with an l''-leader and L'_2 with a sleeping leader f_1 (it can also be the case that L'_2 is just a single isolated f_0, in case L_2 consisted only of 2 nodes). Then l'' waits to meet its q'_2 neighbor to convert it to q_2 and update itself to l. This completes the operation of a line growing one step towards another line and making the other line sleep. A sleeping line cannot increase any more and only loses nodes to lines that are still awake by a similar operation as the one just described. A single leader is guaranteed to always win and this occurs quite fast. Then the unique leader does not need much time to collect all nodes from the sleeping lines to its own line and make the latter spanning.

Theorem 3 ([MS16b]). *Protocol* Fast-Global-Line *constructs a spanning line. It uses 9 states and its expected running time under the uniform random scheduler is* $O(n^3)$.

A variant that backtracks many "sleeping" lines in parallel, is an immediate improvement of *Fast-Global-Line*. The improvement is due to the fact that instead of having the awake leader backtrack sleeping lines node-by-node, we now have any sleeping line backtrack itself, so that many backtrackings occur in parallel. We have some first experimental evidence showing a small improvement in the running time [ALMS15], but we do not yet have a proof of whether this is also an asymptotic improvement. For example, is it the case that the running time of this improvement is $O(n^3/\log n)$ (or even smaller)? This question is open.

Table 1 summarizes a variety of protocols and the corresponding upper and lower bounds that are known for several basic construction problems [MS16b].

4 Generic Constructors

An immediate next question is whether there is a generic constructor capable of constructing a large class of networks. In [MS16b], we answered this in the affirmative by presenting constructors that simulate a TM. The idea is to program the nodes to organize themselves into a network that can serve as a memory of size $O(n^2)$, which is asymptotically maximum and can only be achieved by exploiting the presence or absence of bonds between nodes as the bits of the memory (if only the nodes' local space was used, then the total memory could not exceed $O(n)$). Then the population draws a random network and simulates on the distributed memory a TM that decides whether the network belongs to the target ones. If yes, the population stabilizes to it, otherwise the random experiment and the simulation are repeated (see Figure 3). What makes the construction intricate is that all the sub-routines have to be executed in parallel and potential errors due to this to be corrected by global resets throughout the course of the protocol. This is summarized in the following theorem.

Protocol	# states	Expected Time	Lower Bound
Simple-Global-Line	5	$\Omega(n^4)$ and $O(n^5)$	$\Omega(n^2)$
Fast-Global-Line	9	$O(n^3)$	$\Omega(n^2)$
Cycle-Cover	3	$\Theta(n^2)$ (opt.)	$\Omega(n^2)$
Global-Star	2 (opt.)	$\Theta(n^2 \log n)$ (opt.)	$\Omega(n^2 \log n)$
Global-Ring	9		$\Omega(n^2)$
2RC	6		$\Omega(n \log n)$
kRC	$2(k+1)$		$\Omega(n \log n)$
c-Cliques	$5c-3$		$\Omega(n \log n)$
Graph-Replication	12	$\Theta(n^4 \log n)$	

Table 1. Some established upper and lower bounds [MS16b]. *kRC* (standing for *k-regular connected*) protocol solves a generalization of global ring in which every node has degree $k \geq 2$, *c-cliques* partitions the processes into $\lfloor n/c \rfloor$ cliques of order c each, and *Graph-Replication* constructs a copy of a given input graph.

Theorem 4 (Linear Waste-Two Thirds [MS16b]). DGS$(O(n^2)+O(n)) \subseteq$ **PREL**$(\lfloor n/3 \rfloor)$. *In words, for every graph language L that is decidable by a $(O(n^2)+O(n))$-space TM, there is a protocol that constructs L equiprobably with useful space $\lfloor n/3 \rfloor$.*

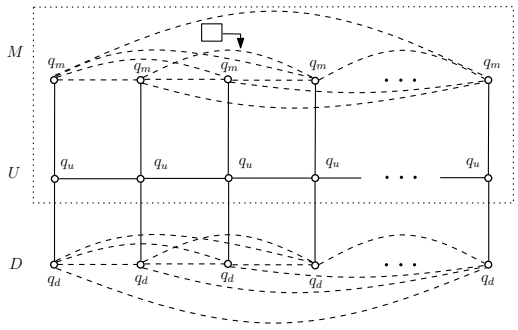

Fig. 3. A partitioning into three equal sets U, D, and M. The line of set U plays the role of an ordering that will be exploited both by the random graph drawing process and by the TM-simulation. The line of set U uses the $\Theta(n^2)$ memory of set M as the memory of the TM. Set D is the useful space on which the output-network will be constructed. Sets U and M constitute the waste.

5 Network Transformations

We shall now consider minimal strengthenings of network constructors that can maximize their computational power, also rendering them capable to terminate.

To this end, we now assume that the initial configuration of the edges can be any configuration in which the active edges form *a connected graph spanning the set of processes*. This choice is motivated by the fact that, without some sort of initial connectivity (or bounded disconnectivity) we can only hope for global computations and constructions that are *eventually stabilizing* (and not *terminating*), roughly because a component can guess neither the number of components not encountered yet nor an upper bound on the time needed to interact with another one of them. [4] The initial configuration of the nodes is either, again, the one in which all nodes are initially in the same state, q_0, or (if needed) the one in which all nodes begin from q_0 apart from a pre-elected unique leader that begins from a distinct initial leader-state l. Unfortunately, even with the additional assumption of bounded initial disconnectivity, it can be proved that non-trivial terminating computation is still impossible.

We now add to the picture a very minimal and natural, but extremely powerful, additional assumption that, combined with our assumptions so far, will lead us to a stronger model. In particular, we equip the nodes with the ability to detect some small local degrees. For a concrete example, assume that a node can detect when its active degree is equal to 0 (otherwise it only knows that its degree is at least 1). A first immediate gain, is that we can now directly simulate any constructor that assumes an empty initial network (like those presented in the previous section): every node initially deactivates the active edges incident to it until its local active degree becomes for the first time 0, and only when this occurs the node starts participating in the simulation.

Our main finding in [MS16a], was that the initial connectivity guarantee together with the ability to modify the network and to detect small local degrees (combined with either a pre-elected leader or a natural mechanism that allows two nodes to tell whether they have a neighbor in common), are sufficient to obtain the *maximum computational power* that one can hope for in this family of models. In particular, the resulting model can compute *with termination* any symmetric predicate [5] computable by a *TM of space* $\Theta(n^2)$, and no more than this, i.e., it is an *exact characterization*. The symmetricity restriction can only be dropped by UIDs or by any other means of knowing and maintaining an ordering of the nodes' inputs. This power is maximal because the distributed space of the system is $\Theta(n^2)$, so we cannot hope for computations exploiting more space. The substantial improvement is that the universal computations are now *terminating* and not just *eventually stabilizing*. It is interesting to point out that the additional assumptions and mechanisms are minimal, in the sense that the removal of each one of them leads to either an impossibility of termination or to a substantial decrease in computational power.

[4] Alternative ways to overcome this are to assume that the nodes know some upper bound on this time [MS15], or, as we shall discuss in the next section, to assume a uniform random scheduler and a unique leader and restrict correctness to be w.h.p..

[5] Essentially, a predicate in this type of models is called *symmetric* (or *commutative*) if permuting the input symbols does not affect the predicate's outcome. This restriction is imposed by the fact that, in general, the nodes cannot be distinguished initially.

The approach to arriving at the above characterization is to develop protocols that exploit the knowledge of the initial connectivity of the active topology and try to transform it to a less symmetric and detectable active topology, without ever breaking its connectivity. The knowledge of initial connectivity and its preservation throughout the transformation process, ensure that the protocol always has all nodes of the network in a single component. Still, if the target-network is symmetric, then there might be no way for the transformation to determine when it has managed to form the network. Instead, the protocols *transform any spanning connected initial topology into a spanning line while preserving connectivity throughout the transformation process*. The spanning line has the advantage that it can be detected under the minimal assumption that a node can detect whether its local degree is in $\{1, 2\}$ and that it is minimally symmetric and, therefore, capable of serving as a linear memory. Preservation of connectivity allows the protocol to be certain that the spanning line contains all processes. So, the protocol can detect the formation of the spanning line and then count (on the $O(\log n)$ cells, i.e., the nodes, of the linear distributed memory) the size of the system. Then the protocol can use the spanning line as it is, for simulating (on the nodes of the line) TMs of space $\Theta(n)$. Going one step further, it is not hard for a protocol to exploit all this obtained information and perform a final transformation that increases the simulation space to $\Theta(n^2)$ (in the spirit of the universal construction of the previous section).

In particular, given an initially connected active topology and the ability of the protocol to transform it, the following set of results can be proved [MS16a]:

- The running time of any protocol that transforms any initial active topology to a spanning line and terminates is $\Omega(n^2 \log n)$.
- If there is a unique leader and a node can detect whether its degree is equal to 1, then there is a time-optimal protocol, with running time $\Theta(n^2 \log n)$ (now defined as the maximum/worst-case expected running time over all possible initial active topologies), that transforms any initial active topology to a spanning line and terminates. This implies a full-power TM simulation as described above.
- If all nodes are initially identical (and even if small local degrees can be detected) then there is no protocol that can transform any initial active topology to an acyclic topology without ever breaking connectivity. The impossibility result is quite strong, proving that, for any initial topology G, there is an infinite family $\mathcal{G}$, such that if the protocol makes G acyclic then it disconnects every $G' \in \mathcal{G}$ in $\Theta(|V(G')|)$ parts. The latter implies that it is impossible to transform to a spanning line with termination.
- There is a plausible additional strengthening that allows the problem to become solvable with initially identical nodes. In particular, the assumption that two interacting nodes can tell whether they have a neighbor in common (*common neighbor detection* mechanism). It can be proved that, with this additional assumption, initially identical nodes can transform any connected spanning initial active topology to a spanning line and terminate in time $O(n^3)$. This implies a full-power TM simulation as described above.

We now describe the aforementioned time-optimal protocol for the simplest case in which there is initially a pre-elected unique leader that handles the transformation. Recall that the initial active topology is connected and the goal is for the protocol to transform the active topology to a spanning line and when this occurs to detect it and terminate (called the Terminating Line Transformation problem). Ideally, the transformation should preserve connectivity of the active topology during its course (or break connectivity in a controlled way). The minimal additional assumption to make the problem solvable, is that a node can detect whether it has local degree 1 or 2 (otherwise it knows that it has degree in $\{0, 3, 4, ..., n-1\}$ without being able to tell its precise value).

Line-Around-a-Star. There is initially a unique leader in state l and all other nodes are in state q_0. Nodes can detect when their degree is 1.

The leader starts connecting with the q_0s (by activating the connection between them in case it was inactive and by preserving it in case it was already active) and converts them to p' trying to form a star with itself at the center. When two p's interact, if the edge is active they deactivate it, trying to become the peripherals of the star. Additionally, if after such a deactivation the degree of a p' is 1, then the p' becomes p to represent the fact that it is now connected only to the leader and has become a normal peripheral. The same occurs if after the interaction of the leader with a q_0, the degree of the q_0 is 1, i.e., the q_0 immediately becomes a normal peripheral p.

When the leader first encounters a p, it starts constructing a line which has as its "left" endpoint the center of the star and that will start expanding over the peripherals until it covers them all. Whenever the leader interacts with an internal node of the line, it disconnects from it (but it never disconnects from the second node of the line, counting from the center; to ensure this, the protocol has that node in a distinguished state i' while all other internal nodes of the line are in state i). The protocol terminates when the degree of the center becomes 1 for the first time (note that it could be 1 also at the very beginning of the protocol but this early termination can be trivially avoided).

Theorem 5 ([MS16a]). *By assuming a pre-elected unique leader and the ability to detect local degree 1, Protocol* Line-Around-a-Star *solves the Terminating Line Transformation problem. Its running time is* $\Theta(n^2 \log n)$*, which is optimal.*

For correctness, observe that every q_0 eventually becomes p, because the center forever attracts the q_0s making them p' and a p' only disconnects from other peripherals until it becomes p. This implies that eventually each non-leader node will become available for the line to expand over it and thus the line will eventually become spanning. Also, the protocol never disconnects the topology because it performs only two types of edge eliminations, (p', p') and (center, node $3 \leq i \leq k$ of the line of length k), which cannot lead to disconnection. Finally it can be shown that the protocol terminates iff the active topology has become a spanning line, by showing that after the line formation subroutine has performed at least on step, the degree of the center first becomes 1 when the active topology becomes a spanning line.

For the running time, the time needed for the leader to connect to every q_0 (and convert all q_0 to p'), is equivalent to the time needed for a particular node to meet every other node, which takes $\Theta(n^2 \log n)$ expected time. Next consider the time for all peripherals to disconnect from one another and become p. If we study this after the time all q_0 have become p', it is the time (in the worst case) needed for all edges to be picked by the scheduler, which takes $\Theta(n^2 \log n)$ expected time. After the completion of both the above, we have a star with the leader at the center and all peripherals are only connected to the leader. Next consider the formation of the line over the peripherals. The right endpoint of the line is always ready for expansion towards another available peripheral. The time needed for the line to cover all peripherals is again the time to meet every other node, therefore takes time $\Theta(n^2 \log n)$ to complete. We finally take into account the time needed for the center to disconnect from the peripherals that are part of the line. We can study this after the line has become spanning. This is simply a star deformation, i.e., the time needed until the center meets all peripherals in order to disconnect from them, taking again time $\Theta(n^2 \log n)$. Putting all these together, we conclude that the running time of the protocol is $\Theta(n^2 \log n)$, which matches the $\Omega(n^2 \log n)$ lower bound mentioned above, therefore the protocol is time-optimal.

6 A Geometric Variant

We shall now discuss a more applied version of network constructors, that may be obtained by adjusting some of the abstract parameters of the general model. In particular, [Mic15] introduced some physical (or geometrical) constraints on the connections that the processes are allowed to form. In the general network constructors model, there were no such imposed restrictions, in the sense that, at any given step, any two processes were candidates for an interaction, independently of their relative positioning in the existing structure/network. For example, even two nodes hidden in the middle of distinct dense components could interact and, additionally, there was no constraint on the number of active connections that a node could form (could be up to the order of the system). This was very convenient for studying the capability of such systems to self-organize into abstract networks and, as we discussed, it helped us show that arbitrarily complex networks are in principle constructible. On the other hand, this is not expected to be the actual mechanism of at least the first potential implementations. First implementations will most probably be characterized by physical and geometrical constraints. To capture this, it was assumed in [Mic15] that each device can connect to other devices only via a very limited (finite and independent of the size of the system) number of ports, usually four or six, which implies that, at any given time, a device has only a bounded number of neighbors. Moreover, the connections are further restricted to be always made at unit distance and to be perpendicular to connections of neighboring ports. Though such a model can no longer form abstract networks, we will see that it is still capable of forming very practical 2D or 3D shapes. This is also in agreement

with natural systems, where the complexity and physical properties of a system are rarely the result of an unrestricted interconnection between entities.

It can be immediately observed that the universal constructors of Section 4 do not apply in this case. In particular, those constructors cannot be adopted in order to characterize the constructive power of the present variant. The reason is that they work by arranging the nodes in a long line and then exploiting the fact that connections are elastic and allow any pair of nodes of the line to interact independently of the distance between them. In contrast, no elasticity is allowed in the more local model that we now consider, where a long line can still be formed, but only adjacent nodes of the line are allowed to interact with each other. As a result, new techniques have to be developed for determining the computational and constructive capabilities of this model. Another main novelty of [Mic15], concerns an alternative approach to overcome the inability of such systems to terminate, by exploiting the ability of nodes to self-assemble into larger structures that can then be used as distributed memories of any desired length and the existence of a uniform random scheduler. Achieving termination is crucial here, as it allows us to develop terminating subroutines that can be sequentially composed to form larger modular protocols. Such protocols are more efficient, more natural, and more amenable to clear proofs of correctness, compared to protocols that are based on composing all subroutines in parallel and "sequentializing" them eventually by perpetual reinitializations (like the one in Section 4).

Now, every node has a bounded number of ports which it uses to interact with other nodes. In the 2D case, there are four ports p_y, p_x, p_{-y}, and p_{-x}, which for notational convenience are usually denoted u, r, d, and l, respectively (for *up*, *right*, *down*, and *left*, respectively). Similarly, in the 3D case there are 6 ports. Neighboring ports are perpendicular to each other, forming local axes. For example, in the 2D case, $u \perp r$, $r \perp d$, $d \perp l$, and $l \perp u$. An important remark is that the above coordinates are only for local purposes and do not necessarily represent the actual orientation of a node in the system. A node may be arbitrarily rotated so that, for example, its x local coordinate is aligned with the y real coordinate of the system or it is not aligned with any real coordinate. Nodes may interact in pairs, whenever a port of one node w is at unit distance and in straight line (w.r.t. to the local axes) from a port of another node v.

The transition function is now of the form $\delta : (Q \times P) \times (Q \times P) \times \{0,1\} \to Q \times Q \times \{0,1\}$, where $P = \{u, r, d, l\}$ ($P = \{p_y, p_z, p_x, p_{-y}, p_{-z}, p_{-x}\}$, respectively, for the 3D case) is the set of *ports*. In every step, a pair of node-ports $(v_1, p_1)(v_2, p_2)$ is selected by an *adversary scheduler* and these nodes interact via the corresponding ports and update their states and the state of the edge joining them according to the transition function δ. A configuration is called *valid*, if any connected component defined by it (when arranged according to the geometrical constraints) is a subnetwork of the *2D grid network with unit distances*. Valid configurations restrict the possible selections of the scheduler at each step. In particular, $(v_1, p_1)(v_2, p_2) \in E_I$ can be selected for interaction (or *is permitted*) at step t iff the configuration that would result after an activation

between (v_1, p_1) and (v_2, p_2) is valid. The interactions are chosen by a uniform random scheduler, which in every step selects independently and uniformly at random one of the permitted interactions. The output shapes of a configuration consist of those nodes that are in output or halting states and those edges between them that are active. We are usually interested in obtaining a single shape as the final output of the protocol. We say that an execution of a protocol on n processes *constructs (stably constructs) a shape* G, if it terminates (stabilizes, resp.) with output G.

The following theorem gives a partial characterization of the constructive power of the 2D version of this model.

Theorem 6 ([Mic15]). *Let $\mathcal{L} = (S_1, S_2, \ldots)$ be a connected 2D shape language, such that $\mathcal{L}$ is TM-computable in space d^2. Then there is a protocol that w.h.p. constructs $\mathcal{L}$. In particular, for all $d \geq 1$, whenever the protocol is executed on a population of size $n = d^2$, w.h.p. it constructs S_d and terminates. In the worst case, the waste is $(d-1)d = O(d^2) = O(n)$.*

The idea is again to organize the population in such a way that it can simulate appropriate TMs; in this case, a type of shape-constructing TMs that will realize their output-shape in the distributed system. Such a TM M constructs a shape on the pixels of a $\sqrt{n} \times \sqrt{n}$ square, which are indexed in a zig-zag way. M takes as input an integer $i \in \{0, 1, \ldots, n-1\}$ and the size n or the dimension $\sqrt{n}$ of the square (all in binary) and decides whether pixel i should belong or not to the final shape, i.e., if it should be *on* or *off*, respectively. In order, to self-organize and simulate the TM, the population first executes a *counting* subroutine, which constructs w.h.p. a line of length $\Theta(\log n)$, containing n in binary. To do this, the protocol requires a pre-elected unique leader. The leader maintains two distributed n-counters and uses them to implement two competing processes, running in parallel. The first process counts the number of nodes that have been encountered once by the leader and the second process counts the number of nodes that have been encountered twice. The game ends when the second counter catches up the first. It can be proved, via a probabilistic analysis of random walks on lines with time and position dependencies, that when this occurs, the leader will almost surely have already counted at least half of the nodes. [6] Then the leader exploits its knowledge of n to construct a $\sqrt{n} \times \sqrt{n}$ square and successfully detect termination of the construction. When it is done, it simulates the TM on the square n distinct times, one for each pixel. As already mentioned, the input to the TM is each time the index of the corresponding pixel and $\sqrt{n}$, in binary, while its output is an *on* or *off* decision for that pixel. Finally, the protocol releases the connected shape consisting of the *on* pixels. It is worth mentioning that it is still open whether the pre-elected leader assumption can be dropped.

[6] In practice, this estimation is expected to be much closer to n than to $n/2$. A first indication is that, in all of our experiments for up to 1000 nodes the estimation was always close to $(9/10)n$ and usually higher.

7 Further Research

An obvious first target is to achieve complete characterizations of the constructible networks both in the basic and in the geometric model. It is also worth noting that existing results on universal construction indicate that the constructive power increases as a function of the available waste. A complete characterization of this dependence would be of special value. Another intriguing question is whether there exists a network constructor for global line that is asymptotically faster than $O(n^3)$. We also do not know yet whether counting the size of the population w.h.p. and with termination is still possible if all nodes are initially identical. Towards refining and extending the existing models, considering hybrid models of active and passive mobility seems interesting. Also, it seems plausible, apart from geometric constraints, to take further physical considerations into account, like mass, strength of bonds, rigid and elastic structure, and collisions. It would also be worth studying structures that optimize some global property or that achieve a desired behavior or functionality. Regarding fault-tolerance capabilities of programmable matter systems, protocols that efficiently reconstruct broken parts of the structure would be of special value. Moreover, we should draw more connections to natural processes and to self-assembly and programmable matter models coming from other research areas (e.g., by comparing the various models via formal simulations). Finally, we believe that more real systems of collectives of large numbers of simple interacting entities (e.g., devices) are needed in order to inspire theory and highlight the feasible mechanisms and, thus, the realistic modeling assumptions.

References

[AAD+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18[4]:235–253, March 2006.

[ALMS15] D. Amaxilatis, M. Logaras, O. Michail, and P. G. Spirakis. NETCS: A new simulator of population protocols and network constructors. *arXiv preprint arXiv:1508.06731*, 2015.

[CKS+12] S. Chatrchyan, V. Khachatryan, A. M. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan, et al. Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716[1]:30–61, 2012.

[DDG+14] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: amoebot–a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 220–222, 2014.

[DGP+16] Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. In *International Conference on DNA-Based Computers*, pages 148–164. Springer, 2016.

[Dot12] D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55:78–88, 2012.

[Dot14] D. Doty. Timing in chemical reaction networks. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–784, 2014.

[GCM05] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. Programmable matter. *Computer*, 38[6]:99–101, 2005.

[GKR10] K. Gilpin, A. Knaian, and D. Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2485–2492. IEEE, 2010.

[KCL+12] A. N. Knaian, K. C. Cheung, M. B. Lobovsky, A. J. Oines, P. Schmidt-Neilsen, and N. A. Gershenfeld. The milli-motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1447–1453. IEEE, 2012.

[KSM+12] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival Jr, N. Assad-Garcia, J. I. Glass, and M. W. Covert. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150[2]:389–401, 2012.

[MCS11] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Mediated population protocols. *Theoretical Computer Science*, 412[22]:2434–2450, May 2011.

[Mic15] O. Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 37–46. ACM, 2015.

[MS15] O. Michail and P. G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81:1–10, 2015.

[MS16a] O. Michail and P. G. Spirakis. Connectivity preserving network transformers. *Theoretical Computer Science*, 2016.

[MS16b] O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29[3]:207–237, 2016.

[RCN14] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345[6198]:795–799, 2014.

[Rot06] P. W. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440[7082]:297–302, 2006.

[RW00] P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 459–468, 2000.

[WCG+13] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 353–354. ACM, 2013.

[Win98] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

[Zak07] M. Zakin. The next revolution in materials. *DARPA's 25th Systems and Technology Symposium (DARPATech)*, 2007.